Introduction to TensorFlow

Olamilekan Wahab, November 2019

Plan

- Why TensorFlow
- Basic Code Structure
- Example: Learning Word Embeddings with Skip-gram
- Variable and Name Scopes
- Visualization with TensorBoard

Plan

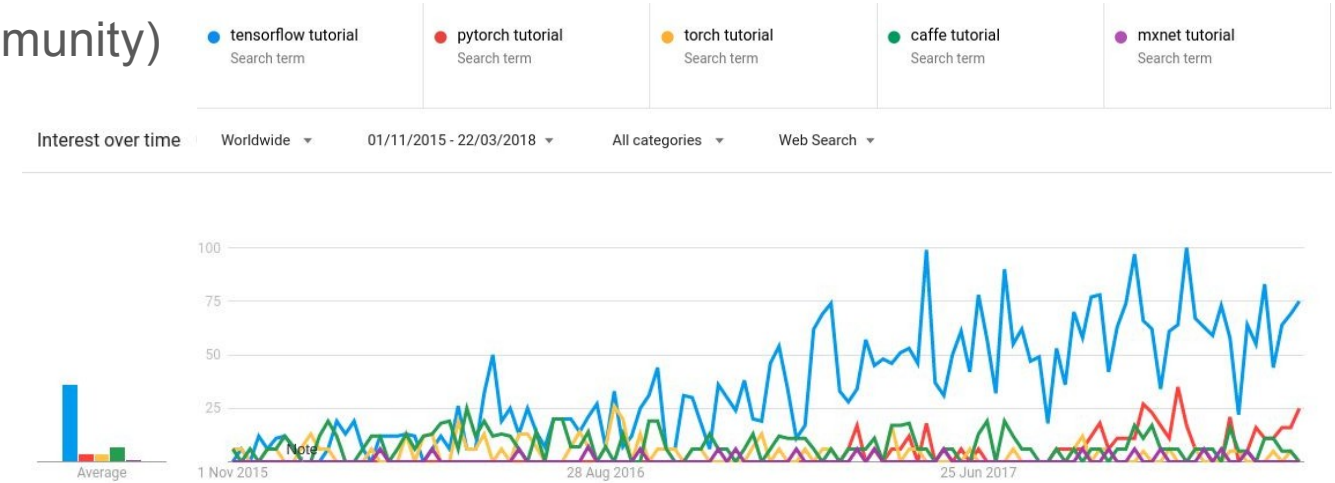
- Why TensorFlow
- Basic Code Structure
- Example: Learning Word Embeddings with Skip-gram
- Variable and Name Scopes
- Visualization with TensorBoard

Goals

- Understand the basic structure of a TensorFlow program
- Be familiar with the main code components
- Understand how to assemble them to train a neural model

Why TensorFlow

- TensorFlow™ is an open source software library for numerical computation using data flow graphs.”
- One of many frameworks for deep learning computations
- Scalable and flexible
- Popular (= big community)



Tensorflow Levels

Primitive	Keras	Layers
lowest, finest control and most flexible Suitable for most machine learning and deep learning algorithms.	highest, most convenient to use, lack flexibility	Somewhere between Primitive and Keras

TensorFlow vs. Numpy

- Few people make this comparison, but TensorFlow and Numpy are quite similar. (Both are N-d array libraries!)

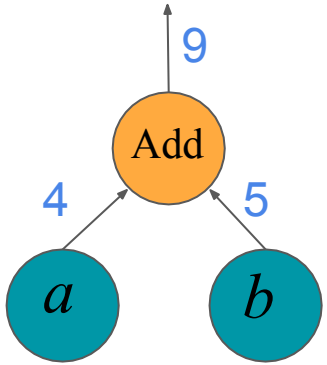
  Numpy has Ndarray support, but doesn't offer methods to create tensor functions and automatically compute derivatives (+ no GPU support).

Basic Code Structure

- View functions as computational graphs
- First build a computational **graph**, and then use a **session** to execute operations in the graph
- This is the basic approach, there is also a dynamic approach implemented in the recently introduced eager mode

Basic Code Structure - Graphs

- Nodes are operators (ops), variables, and constants
- Edges are tensors
  - 0-d is a scalar
  - 1-d is a vector
  - 2-d is a matrix
  - Etc.
- TensorFlow = Tensor + Flow = Data + Flow



But what's a Tensor?

Formally, tensors are multilinear maps from vector spaces to the real numbers (vector space, and dual space)

$$f : \underbrace{V^* \times \cdots V^*}_{p \text{ copies}} \times \underbrace{V \times \cdots V}_{q \text{ copies}} \to \mathbb{R}$$

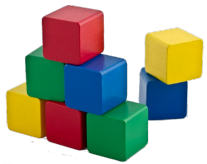
- A scalar is a tensor ($f : \mathbb{R} \to \mathbb{R},\ f(e_1) = c$)
- A vector is a tensor ($f : \mathbb{R}^n \to \mathbb{R},\ f(e_i) = v_i$)
- A matrix is a tensor ($f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R},\ f(e_i, e_j) = A_{ij}$)
- Common to have fixed basis, **so a tensor can be**

Basic Code Structure - Graphs

- **Constants** are fixed value tensors - not trainable
- **Variables** are tensors initialized in a session - trainable

- **Placeholders** are tensors of values that are unknown during the graph construction, but passed as input during a session

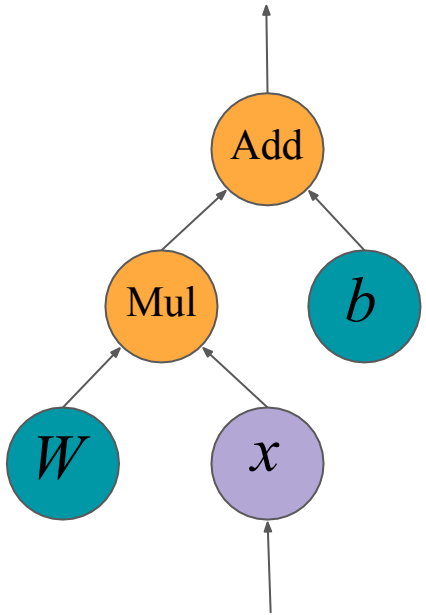
- **Ops** are functions on tensors



Basic Code Structure - Graphs

$$\hat{y} = Wx + b$$

variable ops placeholder variable



Basic Code Structure - Sessions

- Session is the runtime environment of a graph, where operations are executed, and tensors are evaluated

```
>>> import tensorflow as tf
>>> a = tf.constant(4)
>>> b = tf.constant(5)
>>> add_op = tf.add(a, b)
>>> print(add_op)
Tensor("Add:0", shape=(), dtype=int32)
```

```
>>> import tensorflow as tf
>>> a = tf.constant(4)
>>> b = tf.constant(5)
>>> add_op = tf.add(a, b)
>>> with tf.Session() as session:
...     print(session.run(add_op))
...
9
```

- `a.eval()` is equivalent to `session.run(a)`, but in general, "eval" is limited to executions of a single op and ops that returns a value

Basic Code Structure - Sessions

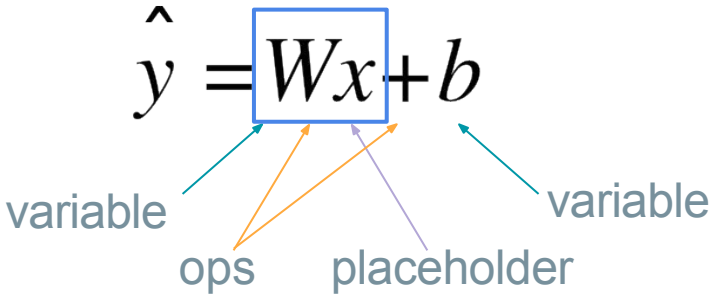
- Session is the runtime environment of a graph, where operations are executed, and tensors are evaluated

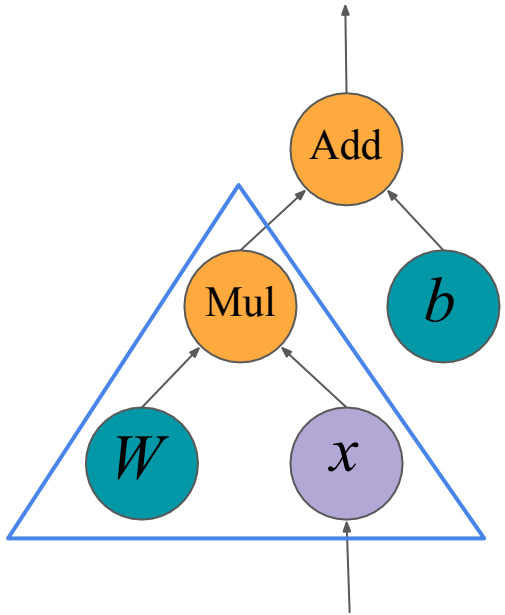
```
>>> import tensorflow as tf
>>> a = tf.constant(4)
>>> b = tf.constant(5)
>>> add_op = tf.add(a, b)
>>> print(add_op)
Tensor("Add:0", shape=(), dtype=int32)
```

```
>>> import tensorflow as tf
>>> a = tf.constant(4)
>>> b = tf.constant(5)
>>> add_op = tf.add(a, b)
>>> with tf.Session() as session:
...     print(session.run(add_op))
...
9
```

- `a.eval()` is equivalent to `session.run(a)`, but in general, "eval" is limited to executions of a single op and ops that returns a value
- Upon op execution, only the subgraph required for calculating its value is evaluated

Basic Code Structure - Sessions



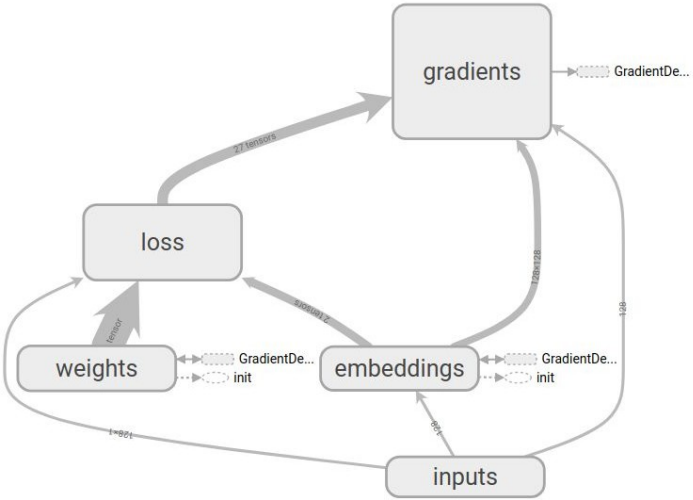


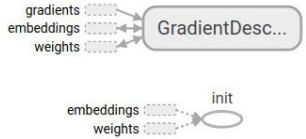
Variable and Name Scopes

- Scopes allow:
  - Grouping of nodes in the graph
  - Sharing variables between graph components
- This is useful as neural networks can become very complex

Variable and Name Scopes

- Scopes allow:
  - Grouping of nodes in the graph
  - Sharing variables between graph components
- This is useful as neural networks can become very complex





Variable and Name Scopes

- `tf.Variable()` creates a new variable under the current scope
- `tf.get_variable()` creates the shared variable if it does not exist yet, or reuse it if it already exists
- The desired behavior is controlled by the current scope

Example from "*Hands-on machine learning with Scikit-Learn and TensorFlow*"

Variable and Name Scopes

- `tf.Variable()` creates a new variable under the current scope
- `tf.get_variable()` creates the shared variable if it does not exist yet, or reuse it if it already exists
- The desired behavior is controlled by the current scope

```
def relu(X, threshold):
    with tf.name_scope("relu"):
        [...]
        return tf.maximum(z, threshold, name="max")

threshold = tf.Variable(0.0, name="threshold")
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = [relu(X, threshold) for i in range(5)]
output = tf.add_n(relus, name="output")
```

Example from *"Hands-on machine learning with Scikit-Learn and TensorFlow"*

Variable and Name Scopes

1

```
with tf.variable_scope("relu"):
    threshold = tf.get_variable("threshold", shape=(),
                                initializer=tf.constant_initializer(0.0))
```

2

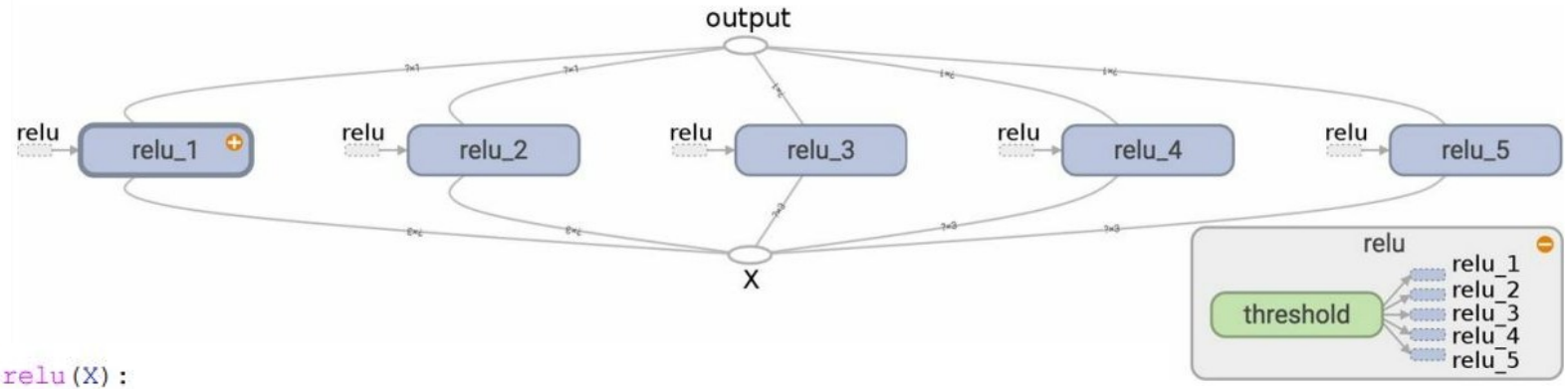
```
with tf.variable_scope("relu", reuse=True):
    threshold = tf.get_variable("threshold")
```

3

```
with tf.variable_scope("relu") as scope:
    scope.reuse_variables()
    threshold = tf.get_variable("threshold")
```

Example from "*Hands-on machine learning with Scikit-Learn and TensorFlow*"

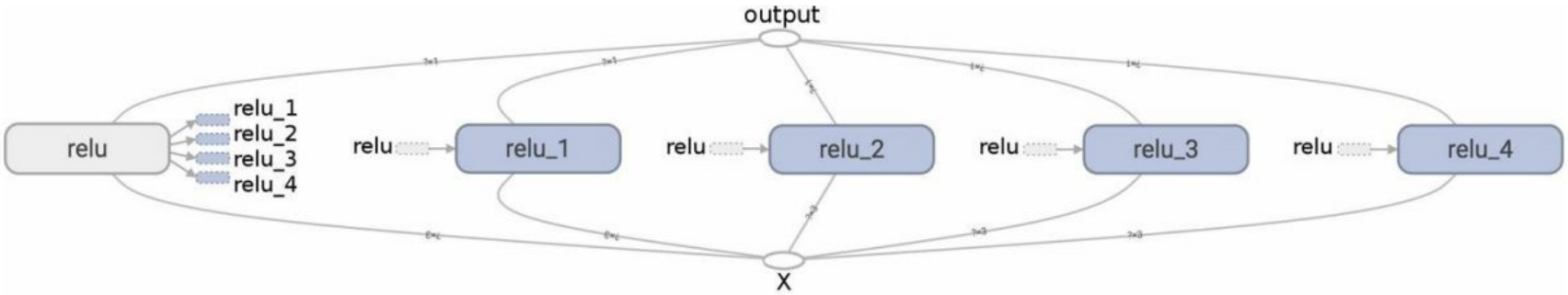
Variable and Name Scopes




```
def relu(X):
    with tf.variable_scope("relu", reuse=True):
        threshold = tf.get_variable("threshold")  # reuse existing variable
        [...]
        return tf.maximum(z, threshold, name="max")

X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
with tf.variable_scope("relu"):  # create the variable
    threshold = tf.get_variable("threshold", shape=(),
                                initializer=tf.constant_initializer(0.0))
relus = [relu(X) for relu_index in range(5)]
output = tf.add_n(relus, name="output")
```

Example from *"Hands-on machine learning with Scikit-Learn and TensorFlow"*

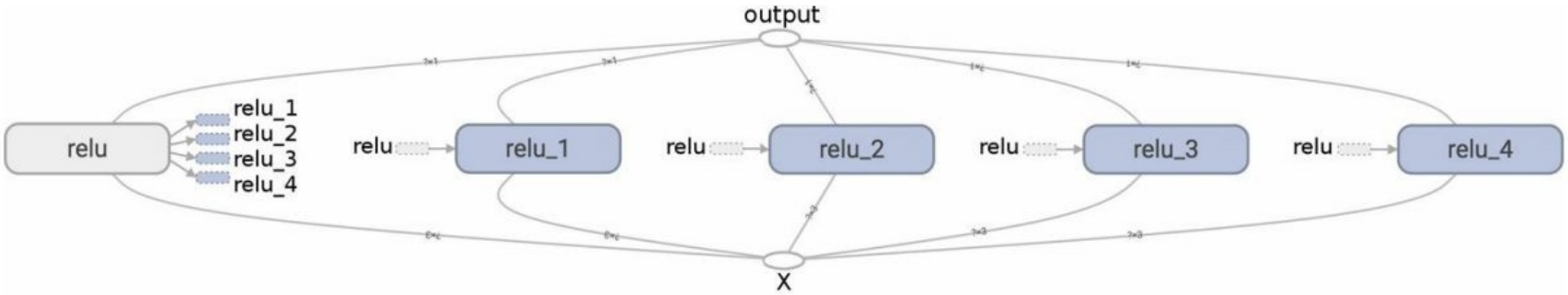
Variable and Name Scopes




```
def relu(X):
    threshold = tf.get_variable("threshold", shape=(),
                                initializer=tf.constant_initializer(0.0))
    [...]
    return tf.maximum(z, threshold, name="max")

X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = []
for relu_index in range(5):
    with tf.variable_scope("relu", reuse=(relu_index >= 1)) as scope:
        relus.append(relu(X))
output = tf.add_n(relus, name="output")
```

Example from *"Hands-on machine learning with Scikit-Learn and TensorFlow"*

Variable and Name Scopes



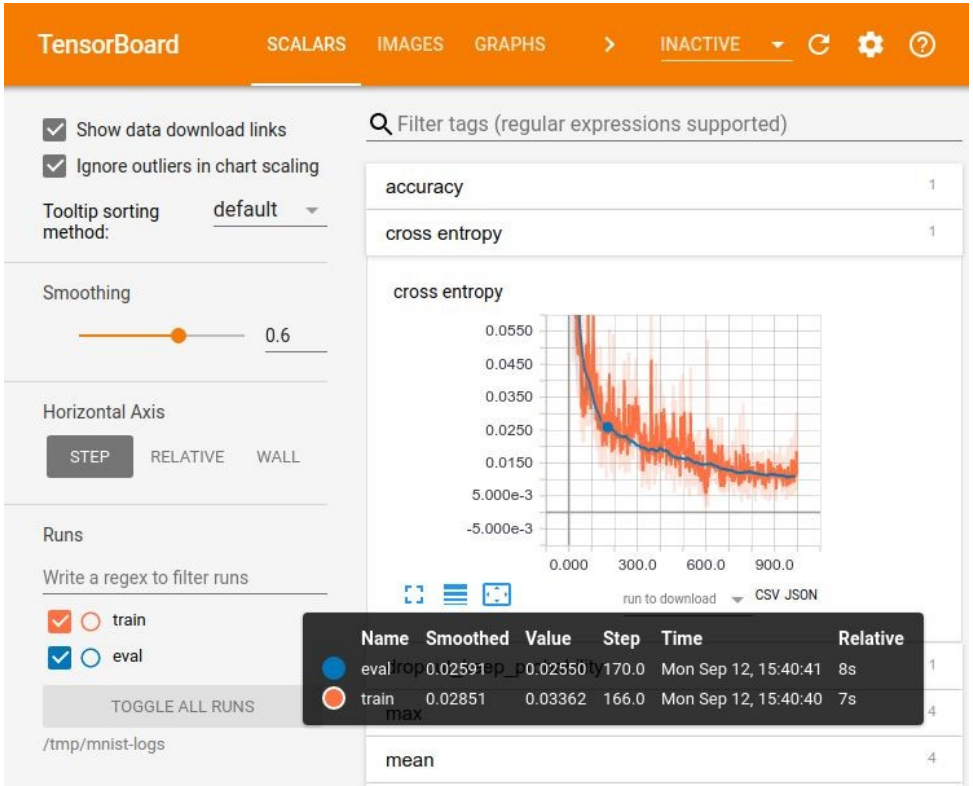

```
def relu(X):
    threshold = tf.get_variable("threshold", shape=(),
                                initializer=tf.constant_initializer(0.0))
    [...]
    return tf.maximum(z, threshold, name="max")

X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = []
for relu_index in range(5):
    with tf.variable_scope("relu", reuse=(relu_index >= 1)) as scope:
        relus.append(relu(X))
output = tf.add_n(relus, name="output")
```

`tf.name_scope` is ignored by `tf.get_variable`

Example from "*Hands-on machine learning with Scikit-Learn and TensorFlow*"

Visualization with TensorBoard

- This is an awesome tool that other frameworks use as well
- It enables browsing the computational graph, monitoring session nodes, and much more



https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

Visualization with TensorBoard - Logging Stats

1. When assembling the graph:
    - Add summary ops
    - Add merge op
2. In a session:
    - Create a file writer
    - Run the merge op every time you want to log stats
    - Add the returned summary to the file writer
3. Load the log to TensorBoard

Resources

- Code & Documentation
  - https://www.tensorflow.org/api_docs/
  - https://github.com/tensorflow
- Tutorials / Courses
  - Tensorflow official tutorials
  - CS 20: Tensorflow for Deep Learning Research
- Books
  - Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. "O'Reilly Media, Inc.", 2017.